A BRANCH AND BOUND ALGORITHM FOR INTEGER AND MIXED-INTEGER LINEAR PROGRAMS

by

Joseph Bannan Missal

United States Naval Postgraduate School

THESIS

A BRANCH AND BOUND ALGORITHM
FOR INTEGER AND MIXED-INTEGER
LINEAR PROGRAMS

by

Joseph Bannan Missal

OCT 1969

A Branch and Bound Algorithm for Integer and Mixed-integer Linear Programs

by

Joseph Bannan Missal
Captain, United States Army
B.S., United States Military Academy, 1964

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
October 1969

ABSTRACT

The algorithm presented is an extension of the Land and Doig branch and bound method combined with the branch selection techniques presented by Beale and Small to solve integer or mixed-integer linear programs. The algorithm obtains the solution by solving a linear program with upper and/or lower bounds on selected branch variables. By systematically changing these bounds, and maintaining only the current canonical form, the solution is assured using a minimum of excess computer storage above that required to solve the linear programming problem. Thus the problem can be solved entirely within the computer core, and the problem converges to the solution faster than most other general integer linear programming algorithms.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES

1. INTRODUCTION

In the literature today, there are many algorithms to solve integer or mixed-integer linear programs, but most of these algorithms are for specialized types of problems, such as transportation problems or machine scheduling problems. As Bellmore and Nemhauser [2] say, "Integer linear programming algorithms are notorious for converging rapidly on one problem and then performing miserably on the next."

The algorithm presented here is Harold Greenberg's natural extension of the Land and Doig branch and bound method [5] , [9] combined with branch selection techniques presented by E.M.L. Beale and R.E. Small [4] , to solve Integer Linear Programs. In addition, the algorithm has been developed to solve mixed-integer linear programs where some, but not all of the variables are integer-restricted.

The problem is solved by primal linear programming using upper and lower bounds on an intelligently selected integer-restricted variable that does not have an integer value in the current solution. These bounds are systematically changed until an optimal solution is assured. In addition to speeding convergence by having less branches than most other methods, this method also uses a minimum of excess computer storage above that required to solve the linear programming problem. Thus the need for offline storage is eliminated and the entire problem is solved and maintained in the core, giving a relatively small cpu time.

Computational results and comparisons with other algorithms are included to show this small time.

II. FORMULATION OF THE PROBLEM

The problem to be solved is: find $X = (x_1,\ldots,x_n)$ that

$$\begin{aligned} &\text{minimizes} && CX' \\ &\text{when} && AX'=B \\ &0 \le x_j \le u_j\ , && j=1,\ldots,n \\ &x_j \text{ integer} && \text{for some or all } j=1,\ldots,n. \end{aligned} \tag{1}$$

C is an n-vector, B is an m-vector, and A is an mxn matrix, and the u_j are positive integers (or are infinite).

To formulate a problem into this form, one need not restrict the slack/surplus variables to being integer, thus there is no need to restrict the elements of the A, B, or C vectors to being integer.

The solution to (1) includes the case where x_j is restricted to being 0 or 1. This accomplished by restricting x_j to being integer with an upper bound of one (i.e. the corresponding u_j equals one).

III. THE BRANCHING PROCEDURE

A. THE LAND AND DOIG METHOD

The branching procedure employed in this algorithm is an extension of the branch and bound method of Land and Doig [9]; their method will be explained first. Land and Doig first solve (1) as a linear program to obtain the continuous solution, $(z^o, x_1^o, \ldots, x_n^o)$. If the continuous solution is all integer, the problem is solved. If there are fractions, this initial value of the objective function, z^o, serves as a lower bound on the value of any feasible optimal integer-solution. One variable, x_j, is then selected whose value is not integer. In the final integer-solution, x_j, must satisfy either $x_j \leq [x_j^o]$, or $x_j \geq [x_j^o] + 1$, where $[x_j^o]$ is the greatest integer less than x_j. The procedure initially involves the solution of two problems: both contain the initial linear programs, but with an additional constraint; in one case the constraint is $x_j = [x_j^o]$, and in the other case it is $x_j = [x_j^o] + 1$. Once these solutions are obtained, and supposing that both are not integer-solutions, the one with the minimum objective value is investigated first, while the other is stored for later investigation. Suppose the minimum of the two objective functions is z^1, with solution:

$(z^1, x_1^1, \ldots, [x_j^o], \ldots, x_n^1)$. Now a new variable, x_k, whose value is not integer, is selected, and three new problems are solved: these are the initial linear program (1) subject to the additional constraints $x_j = [x_j^o]$, $x_k = [x_k^1]$, in the first case, $x_j = [x_j^o]$, $x_k = [x_k^1] + 1$, in the second case, and $x_j = [x_j^o] - 1$, in the third case, all assuming $[x_j] \neq 0$.

The procedure is again repeated until an integer-solution is obtained. At every step where there is a new variable chosen for restriction, there is an additional problem to solve. Thus the next step would have four problems to solve, and the one after that, five, if the integer-solution is not obtained. Once an integer-solution is achieved, its objective function value, z^q, becomes a new upper bound on the optimal solution. With this bound, the procedure then investigates the other problems that are solved at this step. At any point in the branching, when the current upper bound is exceeded, or when an integer-solution is obtained, that branch is terminated.

This complete process enumerates the possibilities for the solution to (1) in a directed tree. The rooted node corresponds to z^o and it has directed branches to the nodes corresponding to the solution of the linear program (1), with x_j^o set at a specific value, each of these integer values representing a single branch. At each of these nodes, the branching is repeated. The minimum objective value at each node determines the next branch, which limits the potentially large size of the tree.

The problems that are stored for further investigation rapidly create an immense storage problem, thus the Land and Doig method becomes unwieldy for problems with more than just a few variables.

The algorithm presented here takes the initial thought of the Land and Doig method, but uses an inequality constraint for the selected variable. This inequality may be as an upper or a lower bound for the variable. By employing a bounded variable linear programming solution technique similar to that from Dantzig [3], the algorithm allows the problem to be solved with a minimum of excess storage used above that storage required to solve the linear programming problem.

B. THE NEW BRANCHING PROCEDURE

As with the Land and Doig method, the procedure obtains the continuous solution to the problem. If this solution is not an integer solution, the procedure then selects a non-integer valued variable, x_j, to branch upon. This creates two new problems; both contain the original linear program (1), but one has the added constraint of $x_j \leq [x_j^o]$, while the other has the added constraint of $x_j \geq [x_j^o] + 1$. Initially, only one of these problems is solved. The solution to this problem is treated in the same manner, and the process is continued until an integer solution is obtained, or the problem becomes infeasible. One thing to notice is at every node there are only two possible

branches, and there are no problems saved. This limiting of branches, alone, tends to reduce the number of nodes required for the solution of the problem. In addition, once an integer solution is obtained, it serves as an upper bound on the final solution, and no branching continues beyond a node where the objective value excedes the current upper bound solution.

The strategy used is to investigate a single branch of the tree until $z^q \geq zz$, where zz is the current upper bound on the optimal solution. At this point, backtracking begins, to find a node that has only one previous branching. This backtracking is accomplished simply by removing all bounds from the nodes that are further out on the tree, and solving the resulting problem. This solution is the re-creation of a previous solution that would have been stored otherwise. The second branch is then made utilizing the initial problem at that node which is re-created from the latest transformation of the equations, and the constraint which was initially ignored at that node, and solving this linear program. At times, the procedure may impose further bounds on a previously bounded variable. This presents no problem since the initial bounds will be satisfied.

Thus, this procedure stores only the current solution and the current bounds. The optimal solution is the existing upper bound solution when it becomes impossible to branch any further. This is realized when backtracking to the rooted node a second time during the solution procedure.

IV. SIMPLEX PROCEDURES USED

The program utilized the two phase simplex method to solve (1), together with a bounded variable technique similar to that in [3]. By using this technique, the problem maintains m constraint equations throughout the solution procedure and has a solution where some of the variables will be at their upper bounds, some of the variables will be basic, and some of the variables will be at their lower bounds.

At each node in the procedure, we are interested in the solution to (1) which satisifies the new bounds given by $0 \leq \bar{l}_j \leq x_j \leq \bar{u}_j \leq u_j$, $j=1,\ldots,n$, where u_j is the problem upper bound, which may be infinite, $\bar{u}_j$ is the imposed upper bound generated by the branching procedure, and $\bar{l}_j$ is the imposed lower bound, also generated by the branching procedure. After achieving a solution for the x_j which satisifies the constraints and the imposed bounds, we can obtain the following canonical form [3], from the constraint equations in (1):

$$\begin{aligned} x_i \quad &+ \sum_{j=m+1}^{n} y_{ij} x_j \quad = d_i, \quad i=1,\ldots,m, \\ -z \quad &+ \sum_{j=m+1}^{n} c_j \, x_j \quad = -z^o, \end{aligned} \tag{3}$$

where the x_i, $i=1,\ldots,m$, are the basic variables, and the x_j, $j=m+1,\ldots,n$, are the non-basic variables. When there is a feasible solution to this problem (3), with upper and

lower bounds, we can use a variation of the usual simplex method to obtain optimality. This variation is suggested by utilizing a method to improve a feasible solution that is not optimal. To improve this feasible solution, we can increase a non-basic variable x_j at its lower bound when $\bar{c}_j < 0$, or we can decrease a non-basic variable x_j at its upper bound when $\bar{c}_j > 0$, where $\bar{c}_j$ is the reduced cost coefficient for the j^{th} variable as found in the objective function in the cononical form. These conditions for optimality come from the following theorem [5]:

Theorem:

any values of x_j, $j=1,\ldots,n$, satisifying the constraint equations in (3) and $0 \leq \bar{l}_j \leq x_j \leq \bar{u}_j \leq u_j$, $j=1,\ldots,n$, are an optimal solution for minimum z, with objective value z_o, iff $\bar{c}_j \geq 0$ for variables at their lower bound, $\bar{l}_j$, and $\bar{c}_j \leq 0$ for variables at their upper bound, $\bar{u}_j$.

The proof is obvious, for any increase in a variable at its lower bound, or any decrease in a variable at its upper bound an only increase z. More detailed proofs may be found in [3], or in [6].

V. THE VARIABLE SELECTION PROCEDURE

While any non-integer variable in the present solution can be chosen for the branching, the size of the directed tree can be further reduced by selecting the one which will produce the optimal solution quickest, thus reducing the number of branchings and nodes. The method employed in this algorithm was developed by Beale and Small [4], using a suggestion of Driebeck.

The process involves comparisons of the marginal cost of increasing and of decreasing each of the non-integer basic variables in the current solution, as in the cost ranging procedure of linear programming, where the marginal cost of changing each of the non-basic variables is considered, and the non-basic variable with the greatest change which still allow feasibility is entered into the basis. The marginal cost, $\mathcal{D}_{Ui}$, of increasing x_i is given by

$$\mathcal{D}_{Ui} = \underset{j, y_{ij} < 0}{\text{Min}} \; (\bar{c}_j / |y_{ij}|),$$

and the marginal cost, $\mathcal{D}_{Di}$, of decreasing x_i is given by

$$\mathcal{D}_{Di} = \underset{j, y_{ij} > 0}{\text{Min}} \; (\bar{c}_j / y_{ij}).$$

where $\bar{c}_j$, is the reduced cost coeficient for the j^{th} variable as found in the objective function in the canonical form. By applying the strategy of taking the variable that makes the worst of the two alternatives as bad as possible, we have the alternative that is the most unlikely to contain the final solution, and we branch on this variable in the

other direction.

This strategy is implimented by computing $\mathcal{L}_{Di} = \mathcal{D}_{Di} \times f_i$ and $\mathcal{L}_{Ui} = \mathcal{D}_{Ui} \times (1 - f_i)$, for each non-integer basic variable that is restricted to being integer. Where f_i is the fractional part of x_i, $x_i = [x_i] + f_i$. The branching is then made on the variable that makes the larger of these two quantities as large as possible. Say that $\underset{i}{\text{Max}}\,\mathcal{L}_{Di} > \underset{i}{\text{Max}}\,\mathcal{L}_{Ui}$, we then go back to find i such that $\mathcal{L}_{Di} = \underset{i}{\text{Max}}\,\mathcal{L}_{Di}$. Thus, the i^{th} basic variable would be the one branched upon, and the direction of the branching is $x_i \geq [x_i] + 1$, since it is the $\mathcal{L}_{Di}$ and not the $\mathcal{L}_{Ui}$ that is the greater.

VI. THE ALGORITHM

A. THE BRANCHING PROCEDURE

The branching procedure is contained in the following algorithm:

1. Definitions

$S(zz,x_1,\ldots,x_n)$ is the current upper bound solution which represents a feasible solution to (1) with objective value zz, and having all variables at integer values which are required to be integer in the solution.

I(L) is the index of the L^{th} bounded variable.

B(L) is the bound restricting the L^{th} bounded variable.

BSIGN(L) indicates whether the bound on the L^{th} bounded variable is an upper or lower bound: BSIGN(L) = + 1 for an upper bound, and BSIGN(L) = - 1, for a lower bound.

N(L) is the counter to show the number of branchings for the L^{th} bounded variable; initially, N(L) = 0.

2. Logic Procedure

a. Set L=0, and take zz as infinite, unless there is some known feasible integer-solution to (1), to get the initial $S(zz,x_1,\ldots,x_n)$. Solve (1) as a linear program to obtain the continuous solution. If this solution has integer values for all of those variables that are integer restricted, the problem is solved. Otherwise go to b,

maintaining the canonical form of the solution to (1).

b. Set L=L + 1. Go to b(1).

(1) Select the basic variable, x_j^o, that is fractional, but required to be integer, which is the most unlikely to produce the optimal solution, and go to b(2).

(2) Set BSIGN(L) opposite the way which will produce the most unlikely chance for the optimal solution and go to b(3) or b(4).

(3) If BSIGN(L) = + 1, set B(L) = x_j^o. Go to c.

(4) If BSIGN(L) = - 1, Set B(L) = x_j^o + 1. Go to c.

c. Solve the linear program using the maintained canonical form with the new bound on the variable x_j, where j = I(L). Go to the following cases, one of which will hold:

(1) If the solution produces an objective value, z, with z > zz, go to d.

(2) If the solution produces an objective value z, with z < zz, and the solution has integers for all of the required variables, redefine $S(zz,x_1,\ldots,x_n)$ as a new feasible integer solution upper bound, where zz = z, $x_1,\ldots,x_n$, is the new solution. Go to d.

(3) If the solution produces an objective value z, with z < zz, and the solution does not have integers for all of the integer-restricted variables, go to b.

(4) If the problem has an infeasible solution, go to d.

d. Set LL = L, and to to d(1).

(1) If N(L)=2, go to d(2). Otherwise N(L)=1; go to d(3).

(2) If L=1, the current feasible solution S(zz,x_1,...,x_n)is optimal, and the algorithm stops. Otherwise, set L=L - 1 and go to d(1).

(3) Solve the linear programming problem starting from the current canonical form with the bounds B(L), B(L + 1),...,B(LL), removed. If j = I(L), and x_j does not become a basic variable, the solution is non-unique; x_j is then made a basic variable. Go to d(4) or d(5).

(4) If BSIGN(L) = +1, set B(L) = B(L) +1, BSIGN(L) = -1, and N(L) = 2, and go to c.

(5) If BSIGN(L) = -1, set B(L) = B(L) -1, BSIGN(L) = +1, and N(L) = 2, then go to c.

This completes the branching part of the algorithm.

B. THE VARIABLE SELECTION PROCEDURE

The selection procedure for which basic variable to branch upon, and which way to branch is contained in the following algorithm.

1. Definitions

I is the i^{th} basic variable.

J is the j^{th} column in the basis.

IB(I) is which problem variable is the I^{th} basic variable.

IMU is the index for which I has the $\max_i \mathcal{L}_{UI}$.

IML is the index for which I has the $\max_i \mathcal{L}_{DI}$.

DU(J) is the current ratio = $|\bar{c}_j/y_{ij}|$, if $y_{ij} < 0$.

DL(J) is the current ratio = $|\bar{c}_j/y_{ij}|$, if $y_{ij} > 0$.

XDU is the current value of the $\min_J$ DU(J).

XDL is the current value of the $\min_J$ DL(J).

XLDU is the current value of the $\max_i \mathcal{L}_{UI}$.

XLDL is the current value of the $\max_i \mathcal{L}_{DI}$.

BSIGN is the indicator for whether the branch requires an upper bound or a lower bound on the variable to be restricted. BSIGN = +1, for an upper bound, and BSIGN = -1, for a lower bound.

M is the number of constraint equations, i.e. the size of the basis.

2. Logic Procedure

a. Set XLDU=XLDL=IMU=IML= 0, I= 0, and go to b.

b. Set I=I + 1. If IB(I) is an artificial variable, or if $x_{IB(I)}$ is not restricted to being integer, or if $x_{IB(I)}$ is integer, go back to a. Otherwise, set J=0, XDU=XDL= infinity, and go to c.

c. Set J = J + 1. If $\bar{c}_j = 0$, repeat step c. If $y_{ij} < 0$, go to c(1), if $y_{ij} > 0$, go to c(2), otherwise, $y_{ij} = 0$, and repeat step c.

(1) DU(J) = $|\bar{c}_j/y_{ij}|$. If DU(J) < XDU, set XDU = DU(J), and go to c(3). Otherwise, go straight to c(3), without setting these values.

(2) DL(J) = $|\bar{c}_j/y_{ij}|$. If DL(J) < XDL, set XDL

= DL(J), and go to c(3). Otherwise, go straight to c(3), without setting these values.

(3) If J < M, go to c, otherwise this is the last comparison for the I^{th} basic variable and go to d.

d. Go to the following cases.

(1) If XDL is still equal to infinity, XLDL is unchanged, otherwise, if XDL x $f_{IB(I)}$ > XLDL, set XLDL = XDL x $f_{IB(I)}$ and set IML = I, go to d(2).

(2) If XDU is still equal to infinity, XLDU is unchanged, otherwise, if XDU x (1 - $f_{IB(I)}$) > XLDU, set XLDU = XDU x (1 -$f_{IB(I)}$), and set IMU = I, go to d(3).

(3) If I > M, go to b, otherwise this is the last basic variable to consider, and go to e.

e. One of the following cases will hold.

(1) If IMU and IML both equal zero, this procedure was called in error, and there are no variables to branch upon.

(2) If XLDU > XLDL, then the variable to branch upon is IMU, and BSIGN = + 1, to set an upper bound on x_{IMU}.

(3) If XLDU < XLDL, then the variable to branch upon is IML, and BSIGN = - 1, to set an lower bound on x_{IML}.

This completes the variable selection portion of the algorithm.

VII. EXAMPLE

The basic branching method, along with the selection procedure for which variable to branch upon are illustrated in the following mixed-integer linear programming problem. This is the same problem used in [1] to illustrate the Land and Doig procedure.

$$\begin{aligned} \text{Min} \quad & 4x_1 + 5x_2 \\ \text{when} \quad & 3x_1 + x_2 - x_3 = 2 \\ & x_1 + 4x_2 - x_4 = 5 \\ & 3x_1 + 2x_2 - x_5 = 7 \\ & x_i \geq 0, \text{ and integer, } i = 1,\ldots,5. \end{aligned} \tag{4}$$

The continuous solution produces the canonical form:

$$\begin{aligned} x_1 + \tfrac{2}{10}x_4 - \tfrac{4}{10}x_5 &= \tfrac{18}{10} \\ x_2 - \tfrac{3}{10}x_4 + \tfrac{1}{10}x_5 &= \tfrac{8}{10} \\ x_3 + \tfrac{3}{10}x_4 - \tfrac{11}{10}x_5 &= \tfrac{42}{10} \\ -z + \tfrac{7}{10}x_4 + \tfrac{11}{10}x_5 &= -\tfrac{112}{10}. \end{aligned} \tag{5}$$

Since the solution has fractions for at least one of the integer restricted variables, we use the selection procedure to tell which variable to branch upon, and which way to branch.

The computations to select the branching variable, and the direction to branch are included for this first branching:

I = 1 $\quad f = 0.8, \quad 1 - f = 0.2.$

J = 1

$\bar{c}_j = 0$, thus no calculation.

J = 2

$DL(2) = \frac{0.7}{0.2} = 3.5,$

$XDL = 3.5.$

J = 3

$DU(3) = \frac{1.1}{0.4} = 2.75,$

$XDU = 2.75.$

$XLDL = 3.5 \times 0.8 = 2.8; \quad IML = 1.$

$XLDU = 2.75 \times 0.2 = 0.55; \quad IMU = 1.$

I = 2 $\quad f = 0.8, \quad 1 - f = 0.2.$

J = 1

$\bar{c}_j = 0$, thus no calculations.

J = 2

$DU(2) = \frac{0.7}{0.3} = 2.33,$

$XDU = 2.33.$

J = 3

$DL(2) = \frac{1.1}{0.1} = 11.0,$

$XDL = 11.0.$

$XDL \times f = 11.0 \times 0.8 = 8.8 > XLDL.$

$XDU \times (1-f) = 2.33 \times 0.2 = 0.467 < XLDU.$

$XLDL = 8.8; \quad IML = 2.$

$XLDU = 0.55; \quad IMU = 1.$

I = 3 $\qquad\qquad$ f = 0.2, $\quad$ 1 - f = 0.8.

J = 1

$\bar{c}_j$ = 0, thus no calculations.

J = 2

$$DL(3) = \frac{0.7}{0.3} = 2.33,$$

XDL = 2.33.

J = 3

$$DU(3) = \frac{1.1}{1.1} = 1.0,$$

XDU = 1.0.

XDL x f = 2.33 x 0.2 = 0.467 < XLDL.

XDU x (1-f) = 1.0 x 0.8 = 0.8 > XLDU.

XLDL = 8.8; $\quad$ IML = 2.

XLDU = 0.8; $\quad$ IMU = 3.

XLDL = 8.8 > 0.8 = XLDU,

therefore: $\quad$ BSIGN = - 1,

IM = IML = 2.

Thus we impose the lower bound on $x_2 \geq 1$, and solve the problem (5) with this added constraint to get the canonical form for node 1:

$$\begin{aligned}
x_1 + \tfrac{2}{3}x_2 - \tfrac{1}{3}x_5 &= -\tfrac{7}{3} \\
x_4 + 2x_2 - \tfrac{1}{3}x_5 &= -\tfrac{8}{3} \\
x_3 - x_2 - x_5 &= 5 \qquad (6) \\
-z + \tfrac{7}{3}x_2 + \tfrac{4}{3}x_5 &= -\tfrac{28}{3},
\end{aligned}$$

with $(z,x_1,x_2,x_3,x_4,x_5) = (35/3,5/3,1,4,2/3,0)$.

Since some of the integer-restricted variables are still fractional, we again use the selection procedure to get the new bound $x_1 \geq 2$, and solve the new problem taken from (6) with the added constraint, $x_1 \geq 2$, to get the canonical form for node 2:

$$\begin{aligned} x_5 - 2x_2 - 3x_1 &= -7 \\ x_4 - 4x_2 - x_1 &= -5 \\ x_3 - x_2 - 3x_1 &= -2 \\ -z + 5x_2 + 4x_1 &= 0, \end{aligned} \tag{7}$$

where $(z,x_1,x_2,x_3,x_4,x_5) = (13,2,1,5,1,1)$, which is integer for all of the integer restricted variables, thus, this solution becomes the solution vector, $S(zz,x_1,\ldots,x_n)$. Next the bound $x_1 \geq 2$, is removed, and the simplex procedure is applied to backtrack to the previous node to obtain (6) again. Now we impose the bound $x_1 \leq 1$, and solve (6) with this new constraint to get the canonical form for node 3:

$$\begin{aligned} x_4 + 5x_1 - 2x_5 &= 9 \\ x_2 + \tfrac{3}{2}x_1 - \tfrac{1}{2}x_5 &= \tfrac{7}{2} \\ x_3 - \tfrac{3}{2}x_1 - \tfrac{1}{2}x_5 &= \tfrac{3}{2} \\ -z - \tfrac{7}{2}x_1 + \tfrac{1}{2}x_5 &= -14, \end{aligned} \tag{8}$$

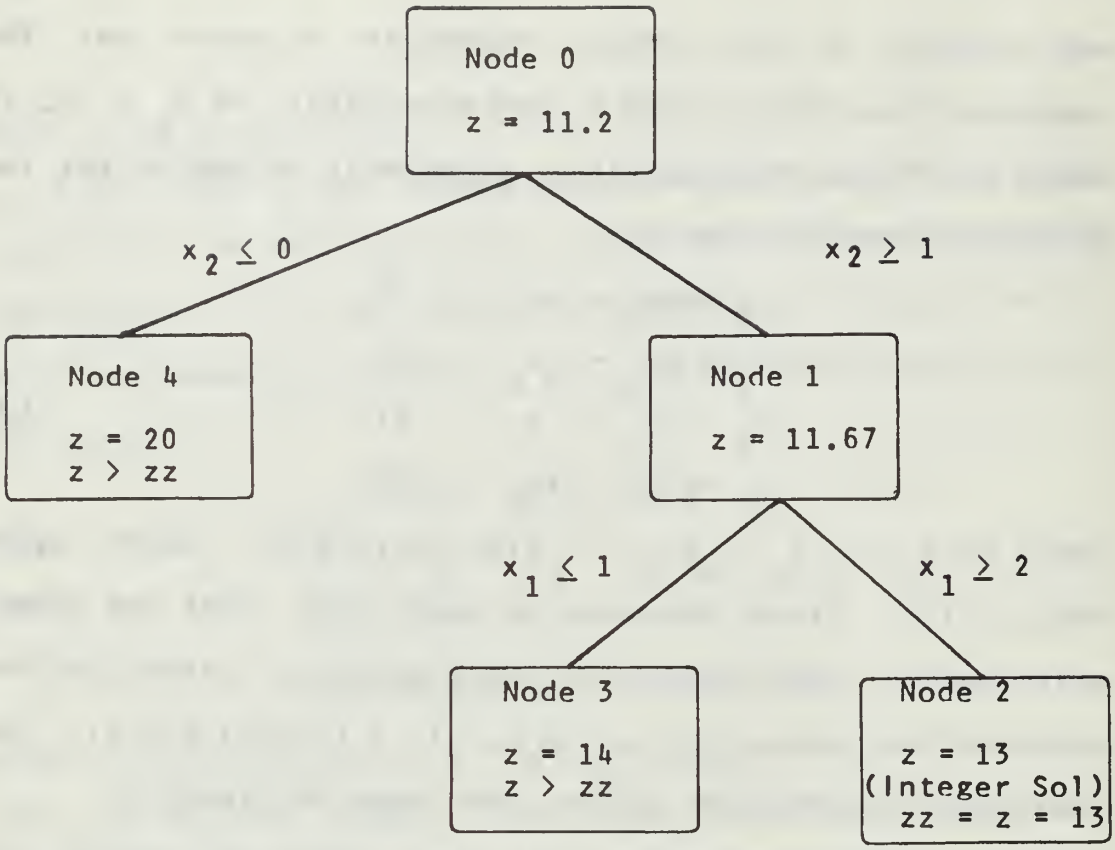
where $(z,x_1,x_2,x_3,x_4,x_5) = (14,1,2,3,4,0)$. Since $z > 13$, we again backtrack to a node where there is only one branch, in this case to node 0, removing the bounds $x_1 \leq 1$, and $x_2 \geq 1$,

and solving by the simplex procedure to again get the canonical form (5). Then a new constraint of $x_2 \leq 0$, is added to (6) and the resulting problem is solved to get the canonical form for node 4:

$$\begin{aligned} x_5 + 10x_2 - 3x_4 &= 8 \\ x_1 + 4x_2 - x_4 &= 5 \\ x_3 + 11x_2 - x_4 &= 13 \\ -z - 11x_2 + 4x_4 &= -20, \end{aligned} \tag{9}$$

where $(z,x_1,x_2,x_3,x_4,x_5) = (20,5,0,13,0,8)$, which again has $z > 13$. Since there are no nodes with just one branch which can be backtracked to, the problem is solved and the solution is: $(zz,x_1,x_2,x_3,x_4,x_5) = (13,2,1,5,1,1)$. The procedure is exhibited in the tree shown in figure 1.

This procedure needed only two nodes to obtain the solution, and four nodes to assure the solution, while this problem (4), as shown in [2] requires five nodes. Thus the advantages of this procedure over the Land and Doig procedure is evident even in this small problem.



Node 0: (112/10,18/10,8/10,42/10,0,0)

Node 1: (35/3,5/3,1,4,2/3,0)

Node 2: (13,2,1,5,1,1)

Node 3: (14,1,2,3,4,0)

Node 4: (20,5,0,13,0,8)

Figure 1

Had we just used some arbitrary rule for selecting which non-integer basic variable to branch upon, the number of nodes may have been much greater. The following tree, in Figure 2, illustrates the arbitrary rule of selecting the first non-integer variable in the basis, x_j, and then branching above this variable by setting the bound, $x_j \geq [x_j] + 1$.

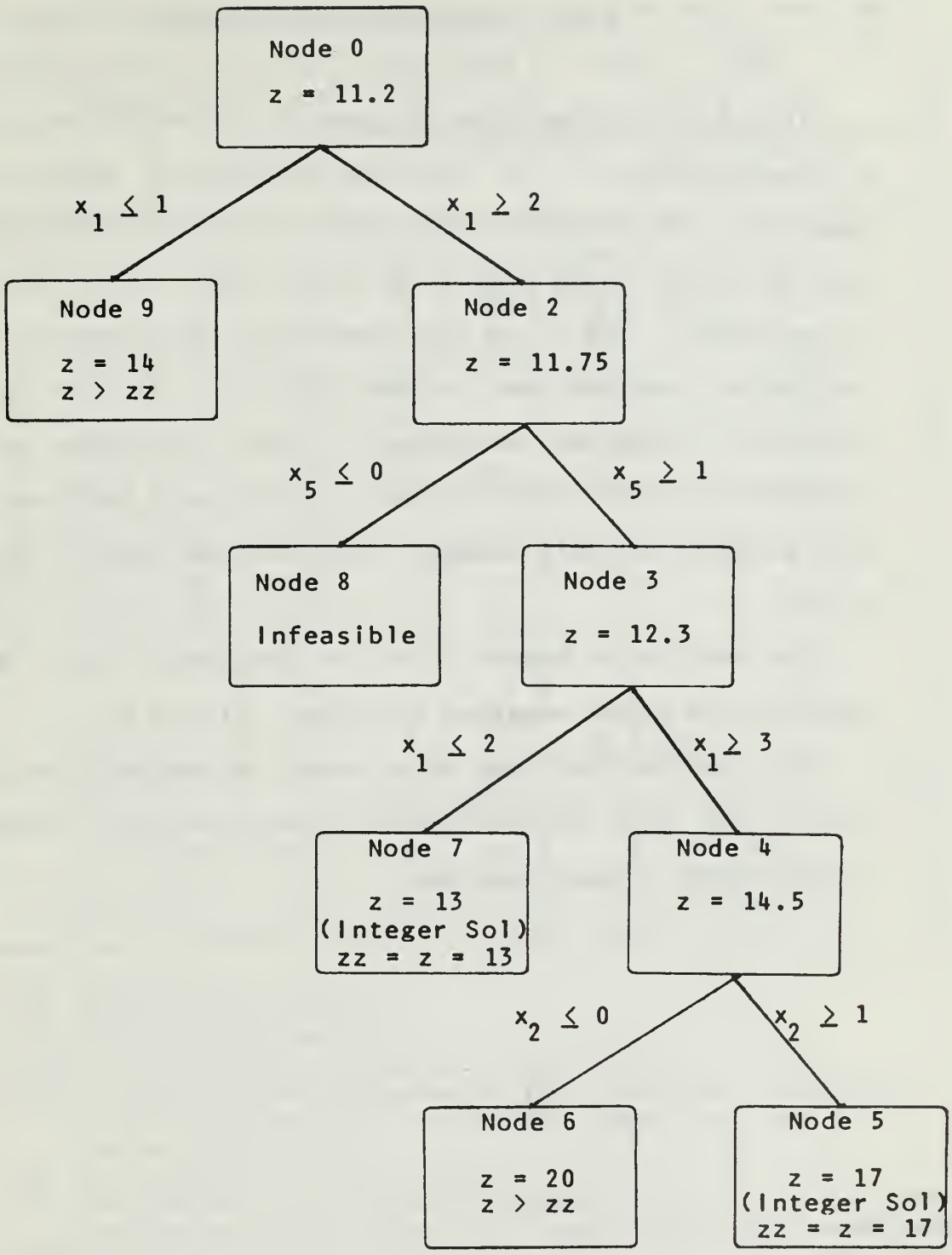

Node 0
z = 11.2
$x_1 \leq 1$
$x_1 \geq 2$
Node 9
z = 14
z > zz
Node 2
z = 11.75
$x_5 \leq 0$
$x_5 \geq 1$
Node 8
Infeasible
Node 3
z = 12.3
$x_1 \leq 2$
$x_1 \geq 3$
Node 7
z = 13
(Integer Sol)
zz = z = 13
Node 4
z = 14.5
$x_2 \leq 0$
$x_2 \geq 1$
Node 6
z = 20
z > zz
Node 5
z = 17
(Integer Sol)
zz = z = 17

Figure 2

VIII. COMPUTATIONAL RESULTS

The algorithm has been programmed in Fortran IV, and run on an IBM 360/67. The complete program is given in the appendix. The computer core needed to run a problem is 62K plus $36n + 32m + 4m \times (m + n)$ bytes, where n is the number of variables, and m is the number of constraints in the particular problem when in the form as (1). The program utilizes a simplex procedure in real variables with a round-off correction which requires that all problems using this program have all integer coefficients when in the form as (1).

The results in Figure 3 are for problems in [7], and the data for the other programs are from [7] and [8].

The results show that this branch and bound algorithm is faster than most of the general algorithms for integer or mixed-integer linear programs.

Comparison of Computational Experience on Haldi and IBM

Source	m (g)	n (h)	LIP1 (a)	IPM2 (b)	IPM3 (c)	BBA (d)		BVA (e)
			Iter (f)	Iter	Iter	Iter	Time (sec)	Time(j) (sec)
Haldi								
1	4	5	25	19	51	94	3.15	--
2	4	5	21	18	58	26	.83	--
3	4	5	22	17	80	41	1.33	--
4	4	5	17	29	35	12	.38	--
5	6	5	181	765	F(i)	22		30
6	6	5	126	20	F	245	8.2	12
7	4	5	173	731	F	31	1.03	24
8	4	5	125	20	F	31	1.02	9
9	6	6	43	39	192	22		8
10	10	12	120	F	F			17
11	21	56	104	--	--			--
12	21	56	103	--	--			--
13	21	56	683	--	--			--
14	21	56	106	--	--			--
15	21	56	877	--	--			--
IBM								
1	7	7	11	7	8			18
2	7	7	30	10	17			21
3	3	4	63	2	13	48	1.7	4
4	15	15	60	24	24			23
5	15	15	332	453	1078			154
6	31	31	341	60	417			--
7	12	50	391	F	74			--
8	12	37	63	453	33			--
9	50	15	957	5561	F			--

(a) Values are from [7].
(b) Values are from [7].
(c) Values are from [7].
(d) The algorithm presented in this paper.
(e) The algorithm discussed in [8], times are from [8].
(f) Iterations are the total pivot steps required to solve the problem.
(g) The number of constraints.
(h) The number of variables, excluding slacks. Each problem has as many slacks as it does constraint equations.
(i) An "F" means that the problem was unsolved after 3000+ iterations.
(j) Times to solve problem on IBM 360 series computers.

Figure 3

```
      COMMON A( 25, 25),B( 25, 25),IB( 25),H( 25),AX( 25)
      COMMON X( 25),XLB( 25),XUB( 25),XI( 25),Y( 25),R( 25)
      COMMON  XSOL( 25),IRV( 25)
      COMMON MM,N,M,JS,E,IR,AD,W,Z,IM,DI,ITR,XH,AH,HH,EE,BND
      COMMON ZZ,DJS,ITER,L,IW,NODEO,INTRES,XT,XK,ID,RL,BSIGN
      COMMON  IREAD,IWRITE,IBNB
      DIMENSION ITITLE(15)
      DATA ISTOP/'STOP'/

C PROGRAM TO ENTER DATA AND CALL LP.

C MM = NUMBER OF CONSTRAINT EQUATIONS.
C N=NUMBER OF VARIABLES--THE # OF ELEMENTS IN THE X-VECTOR.
C BND = 0 MEANS NO BOUNDS.
C BND = 1 MEANS BOUNDS.
C IL IS A CONTROL VARIABLE TO SIGNAL THE END OF DATA FOR A
C PARTICLUAR READ STATEMENT.  IL=1 FOR LAST CARD, 0 OTHER.
C INTRES = 0 MEANS THAT THE PROBLEM IS A ALL INTEGER
C RESTRICTED PROBLEM.
C INTRES = 1 MEANS THAT THE PROBLEM IS A MIXED-INTEGER
C PROBLEM.
C IRV(I) = 0 MEANS THAT THE I-TH VARIABLE IS NOT INTEGER
C RESTRICTED.
C IRV(I) = 1 MEANS THAT THE I-TH VARIABLE IS INTEGER
C RESTRICTED.
C NOTE THAT THE SLACK/SURPLUS VARIABLES DO NOT HAVE TO BE
C INTEGER RESTRICTED.
C WHEN INTRES= 0, THE VALUE OF IRV(I) IS IMMATERIAL, THE
C PROGRAM WILL HAVE ALL IRV(I)=0, BUT WILL DISREGARD THIS.
C M IS THE VARIABLE TO AUGMENT THE MARTIX FOR THE OBJ. FUNC.
C H(I) ARE THE RIGHT HAND SIDES,H(M) = INITIAL Z.
C H(M)=INITIAL Z, WHICH IS ZERO UNLESS OTHERWISE PUT IN PRO.
C A(M,J) ARE THE COSTS.


      IWRITE = 8
      IREAD  = 4
    2 FORMAT (2I5,F5.0,I5,15A4)
    4 FORMAT(I1,2I3,F5.0)
    6 FORMAT (I1,I3,F5.0,I2)
   10 FORMAT(1H ,10F8.0)
   14 FORMAT(1H0,10X, ' HAVE FINISHED '/'1')
   49 FORMAT (1H1,35X,15A4//)
   51 FORMAT (' MINIMIZE:')
   52 FORMAT (' SUBJECT TO:')
   53 FORMAT (1X,24F5.0)
   54 FORMAT ('+',124X,'=',F5.0)
   55 FORMAT (' THIS PROBLEM HAS',I5,' CONSTRAINT EQUATIONS,
     1 ,I5,' VARIABLES, AND NO BOUNDS',/)
   56 FORMAT (' THIS PROBLEM HAS',I5,' CONSTRAINT EQUATIONS,
     1 ,I5,' VARIABLES, AND HAS UPPER BOUNDS',/)
   57 FORMAT(' WITH THE FOLLOWING UPPER BOUNDS ON THE VARI',
     1'BLES (A -1 INDICATES NO BOUND ON THAT VARIABLE):')
   58 FORMAT(' THE FOLLOWING VARIABLES MARKED WITH A ONE A',
     1'RE INTEGER RESTRICTED:')
   59 FORMAT (1X,24I5)
   15 READ (IREAD,2) MM,N,BND,INTRES,ITITLE
      IF (ITITLE(1).EQ.ISTOP)  STOP
      M = MM+1
      DO 1 I=1,M
      H(I)=0.
      DO 1 J=1,N
    1 A(I,J)=0.
      DO 9 J=1,N
      IRV(J) = 0
      XUB(J)=-1.
    9 XLB(J)=0.
    3 READ (IREAD,4)IL,I,J,A(I,J)
      IF(IL.EQ.0)GO TO 3
    5 READ (IREAD,6)IL,I,H(I)
      IF(IL.EQ.0)GO TO 5
```

```
    7 READ (IREAD,6) IL,J,A(M,J),IRV(J)
      IF(IL.EQ.0)GO TO 7
      IF(BND.EQ.0)GO TO 8
   11 READ(IREAD,6) IL,J,XUB(J)
      IF(IL.EQ.0)GO TO 11
    8 WRITE (IWRITE,49) ITITLE
      IF (BND.EQ.0.) WRITE (IWRITE,55) MM,N
      IF (BND.EQ.1.) WRITE (IWRITE,56) MM,N
      WRITE (IWRITE,51)
      WRITE (IWRITE,53) (A(M,J),J=1,N)
      IF (BND.EQ.0) GO TO 61
      WRITE(IWRITE,57)
      WRITE (IWRITE,53)  (XUB(J),J=1,N)
   61 IF (INTRES.EQ.0)  GO TO 62
      WRITE (IWRITE,58)
      WRITE (IWRITE,59)    (IRV(I),I=1,N)
   62 WRITE( IWRITE,52)
      DO 60 L = 1,MM
      WRITE (IWRITE,53) (A(L,J),J=1,N)
   60 WRITE (IWRITE,54)  H(L)
C EXECUTION TIMED FROM THIS POINT
      CALL LP
      WRITE (IWRITE,14)
C END OF EXECUTION TIMING
      WRITE (IWRITE,49) ITITLE
      DO 47 J=1,N
      IF(XSOL(J))48,47,48
   48 WRITE(IWRITE,50)J,XSOL(J)
   50 FORMAT(1HO,5X,5HXSOL(,I2,2H)=,F13.4)
   47 CONTINUE
      WRITE (IWRITE,45)ZZ
   45 FORMAT(1HO,3HZZ=,F10.5)
      GO TO 15
      END


      SUBROUTINE LP
      COMMON A( 25, 25),B( 25, 25),IB( 25),H( 25),AX( 25)
      COMMON X( 25),XLB( 25),XUB( 25),XI( 25),Y( 25),R( 25)
      COMMON  XSOL( 25),IRV( 25)
      COMMON MM,N,M,JS,E,IR,AD,W,Z,IM,DI,ITR,XH,AH,HH,EE,BND
      COMMON ZZ,DJS,ITER,L,IW,NODEO,INTRES,XT,XK,ID,RL,BSIGN
      COMMON  IREAD,IWRITE,IBNB


C THIS IS THE SUBPROGRAM WHICH CONTROLS THE SIMPLEX
C PROCEDURES, THE INITIAL SET-UP OF THE COMPUTATIONAL
C ARRAYS, AND THE TERMINATION OF THE PROBLEM.


      NODEO = 0
      ZZ=1000000.
      IW=0
      Z=H(M)
      L=0
      M=MM+2
      ID=0
      DO 3 J=1,MM
      DO 2 I=1,M
    2 B(I,J)=0.
      B(M,J)=-1.
    3 B(J,J)=1.
C XI(J)=+1. MEANS INCREASE.
C XI(J)=-1. MEANS DECREASE.
C XI(J)=2.  MEANS BASIC.
C XI(J)=0. MEANS NONBASIC.
      DO 24 J=1,N
      XI(J)=0.
   24 X(J)=0.
      DO 4 I=1,MM
      Y(I)=H(I)
    4 IB(I)=-I
```

```
      AD=1.
      DI=1.
      E=.5
      W=0.
      DO 5 J=1,N
      A(M,J)=0.
      DO 5 I=1,MM
    5 A(M,J)=A(M,J)-A(I,J)
      DO 6 I=1,MM
    6 W=W+H(I)
C W = SUM OF THE H(I), I=1,...,MM.
      WRITE(IWRITE,14)W
   14 FORMAT(1HO,5X,' INITIAL W= ',F12.4)
      ITR=C
      ITER=0
    8 JS=0
      IF (ITER.GT.225) WRITE (IWRITE,5100)
 5100 FORMAT(10X,'ITERATIONS ARE ABOVE LIMIT SET IN PROG.')
      IF (ITER.GT.225) RETURN
      DO 9 J=1,N
      IF(XI(J).EQ.2.) GO TO 9
      D=0.
      DO 7 K=1,MM
    7 D=D+B(M,K)*A(K,J)
      IF(W.EQ.0.) D=D+A(M,J)
      IF(ABS(D).LE.E)GO TO 9
      IF(D)15,9,16
   15 IF(XUB(J).EQ.-1.)GO TO 19
      IF(ABS(XUB(J)-X(J)).LT.E)GO TO 9
C INCREASE.
   19 XI(J)=1.
      GO TO 17
   16 IF(ABS(X(J)-XLB(J)).LT.E)GO TO 9
C DECREASE.
      XI(J)=-1.
   17 IF(JS.GT.0)GO TO 10
   61 JS=J
      DJS=D*XI(J)
      GO TO 9
   10 IF(DJS-D*XI(J).GE.E)GO TO 61
    9 CONTINUE
      IF(JS.EQ.0)GO TO 101
      WRITE(IWRITE,213)JS,DJS,XI(JS)
  213 FORMAT(1H ,5X,3HJS=,I5,5X,4HDJS=,F10.4,5X,3HXI=,F10.4)
  221 CALL ELEM
      ITER=ITER+1
      IF (L.LT.C) GO TO 529
      IF(IR.GT.C)GO TO 36
      IF(BND.EQ.0)GO TO 33
      IF (BND.EQ.1.AND.L.EQ.0)  GO TO 33
      IF(XH.EQ.1000000.)GO TO 33
                CALL FLAG (1105)
   36 DO 38 I=1,MM
      IF(IB(I))39,528,40
   39 Y(I)=Y(I)-AX(I)*XH*XI(JS)
      GO TO 38
   40 K=IB(I)
      X(K)=X(K)-AX(I)*XH*XI(JS)
   38 CONTINUE
      X(JS)=X(JS)+XH*XI(JS)
      IF(W.EQ.C.)GO TO 62
      W=W+AX(M)*XH*XI(JS)
   62 Z=Z+AX(MM+1)*XH*XI(JS)
      WRITE(IWRITE,28)W,Z,ITER
   28 FORMAT(1H ,10X,2HW=,F10.4,10X,2HZ=,F10.4,5X,5HITER=,I5
      IF(IR)528,50,44
   44 CALL PIVOT
      CALL FLAG ( 3083  )
   50 IF(IR.EQ.0) XI(JS)=0.0
      IF(W.GT.E)GO TO 8
      IF(IW.EQ.1)GO TO 8
      W=0.
```

```
      M=M-1
      IW=1
      WRITE(IWRITE,73)
   73 FORMAT(1HO,10X,' START PHASE TWO ')
      GO TO 8
   33 WRITE(IWRITE,704)
      WRITE (6,5701)L,R(L),Z,ZZ
  704 FORMAT(1HO,10X,18HUNBOUNDED SOLUTION)
      RETURN
  101 IF(W.GT.E)GO TO 35
      WRITE(IWRITE,204)
  204 FORMAT(1HO,5X,6HRESULT)
      DO 31 J=1,N
      IF(ABS(X(J)).LT.E)GO TO 31
      WRITE(IWRITE,200)J,X(J)
  200 FORMAT(1H ,5X,2HX(,I2,2H)=,F10.4)
   31 CONTINUE
      Z=SIGN(AINT(DI*ABS(Z)+.5)/DI,Z)
      WRITE(IWRITE,527)Z
  527 FORMAT(1HO,5X,2HZ=,F17.5)
C HAVE CONTINUOUS SOLUTION.
      IF (INTRES.EQ.1) GO TO 301
      IF (Z+1-E.GE.ZZ) GO TO 134
  301 IF (Z+E.GE.ZZ)   GO TO 134
      IF(DI-1.)601,601,603
  601 WRITE(IWRITE,701)
      WRITE (6,5701)L,R(L),Z,ZZ
 5701 FORMAT (5X,'R(',I3,') = ',F14.6,5X,'Z = ',F11.4,
     15X,'ZZ = ',F11.4,/)
  701 FORMAT (1HO,5X,22HHAVE INTEGERS REQUIRED)
      GO TO 51
  603 IF (INTRES.EQ.1) GO TO 550
      IF (ABS(ABS(Z)-AINT(ABS(Z)+E)).GE.E) GO TO 6032
  550 DO 637 I=1,MM
      IF (INTRES.EQ.1.AND.IRV(IB(I)).EQ.0) GO TO 637
      IF(IB(I))637,637,83
   83 J=IB(I)
      IF(ABS(X(J)-AINT(ABS(X(J))+E))-E)637,637,6032
  637 CONTINUE
C HAVE ALL INTEGERS REQUIRED.
      WRITE(IWRITE,701)
      WRITE (6,5701)L,R(L),Z,ZZ
   51 DO 45 J=1,N
      XSOL(J) = AINT (X(J) + 0.5)
   45 IF (INTRES.EQ.1.AND.IRV(J).EQ.0)  XSOL(J) = X(J)
      ZZ=Z
      IF (BND.EQ.0)  GO TO 46
      IF (BND.EQ.1.AND.L.EQ.0) GO TO 46
      GO TO 34
 6032 WRITE(IWRITE,703)
      WRITE (6,5701)L,R(L),Z,ZZ
  703 FORMAT(' HAVE FRACTIONS')
      BND=1.
      IF(ID.EQ.500) GO TO 41
   42 CALL BOUND
      GO TO 8
   35 WRITE(IWRITE,205)
      WRITE (6,5701)L,R(L),Z,ZZ
  205 FORMAT(1HO,' INFEASIBLE')
      IF(BND.EQ.0.)GO TO 528
      GO TO 34
  134 WRITE (IWRITE,702)
      WRITE (6,5701) L,R(L),Z,ZZ
  702 FORMAT (' Z VALUE GREATER THAN SOLUTION UPPER BOUND ')
   34 CALL UNBND
      IF (L.EQ.1.AND.NODEO.EQ.2)  GO TO 46
      IF(L.EQ.1.AND.ABS(R(L)).GT.500)  NODEO = 2
      GO TO 8
   41 K=XK
      IF(XI(K).EQ.2.) GO TO 42
      IF(ABS(XT-X(K))-E) 260,210,220
  220 IF(R(L)) 230,8,240
```

```
  240 IF (XT.LE.X(K)) GO TO 250
      JS = K
      XI(K) = +1.
      L = L - 1
      GO TO 221
  250 ID = 0
      XLB(K) = XT
      GO TO 42
  210 ID=0
      XUB(K)=XT
      GO TO 42
  230 IF(XT.GE.X(K)) GO TO 210
      JS=K
      XI(K)=-1.
      L = L - 1
      GO TO 221
  260 IF (R(L))210,8,250
  529 WRITE (6,5701) L,R(L),Z,ZZ
      WRITE (IWRITE,5702)
 5702 FORMAT(' ERROR, L IS NEGATIVE.')
   46 DO 47 J=1,N
      IF(XSOL(J))48,47,48
   48 WRITE(IWRITE,49)J,XSOL(J)
   49 FORMAT(1HC,5X,5HXSOL(,I2,2H)=,F13.4)
   47 CONTINUE
      WRITE (IWRITE,52)ZZ
      WRITE(6,52)ZZ
   52 FORMAT(1HO,3HZZ=,F10.5)
  528 RETURN
      END


      SUBROUTINE AXE
      COMMON A( 25, 25),B( 25, 25),IB( 25),H( 25),AX( 25)
      COMMON X( 25),XLB( 25),XUB( 25),XI( 25),Y( 25),R( 25)
      COMMON  XSOL( 25),IRV( 25)
      COMMON MM,N,M,JS,E,IR,AD,W,Z,IM,DI,ITR,XH,AH,HH,EE,BND
      COMMON ZZ,DJS,ITER,L,IW,NODEO,INTRES,XT,XK,ID,RL,BSIGN
      COMMON  IREAD,IWRITE,IBNB

C THIS IS THE SUBPROGRAM WHICH COMPUTES THE A(JS) COLUMN
C NEEDED TO PERFORM THE PIVOT STEP.

      DO 21 I=1,M
      AX(I)=0.
      DO 21 K=1,MM
      AX(I)=AX(I)+B(I,K)*A(K,JS)
   21 CONTINUE
      AX(MM+1)=AX(MM+1)+A(MM+1,JS)
      RETURN
      END


      SUBROUTINE PIVOT
      COMMON A( 25, 25),B( 25, 25),IB( 25),H( 25),AX( 25)
      COMMON X( 25),XLB( 25),XUB( 25),XI( 25),Y( 25),R( 25)
      COMMON  XSOL( 25),IRV( 25)
      COMMON MM,N,M,JS,E,IR,AD,W,Z,IM,DI,ITR,XH,AH,HH,EE,BND
      COMMON ZZ,DJS,ITER,L,IW,NODEO,INTRES,XT,XK,ID,RL,BSIGN
      COMMON  IREAD,IWRITE,IBNB

C THIS IS THE SUBPROGRAM WHICH DOES THE SIMPLEX PIVOT STEP.

      AD=AD*ABS(AX(IR))
      DI=AINT(DI*ABS(AX(IR))+.5)
   84 WRITE(IWRITE,214)IR,AX(IR)
  214 FORMAT(1HO,25X,3HIR=,I3,3X,3HAX=,F8.4)
      IF(IB(IR))1,1,2
```

```
    2 K=IB(IR)
      XI(K)=0.
    1 IB(IR)=JS
      XI(JS)=2.
      E=.5/DI
      DO 29 I=1,M
      IF(I.EQ.IR)GO TO 29
      P=-AX(I)/AX(IR)
      H(I)=H(I)+P*H(IR)
      DO 31 J=1,MM
   31 B(I,J)=B(I,J)+P*B(IR,J)
   29 CONTINUE
      P=1./AX(IR)
      H(IR)=H(IR)*P
      DO 33 J=1,MM
   33 B(IR,J)=B(IR,J)*P
   91 DO 92 I=1,M
      H(I)=SIGN(AINT(DI*ABS(H(I))+.5)/DI,H(I))
      DO 92 J=1,MM
   92 B(I,J)=SIGN(AINT(DI*ABS(B(I,J))+.5)/DI,B(I,J))
      W = AINT(DI*W+0.5)/DI
      Z = SIGN(AINT(DI*ABS(Z) + 0.5)/DI,Z)
   95 ITR=ITR+1
      WRITE(IWRITE,210)ITR,W,Z,DI,AD
  210 FORMAT(1HO,5X,4HITR=,I5,5X,2HW=,F12.8,5X,2HZ=,F12.5,5X
     1,3X,3HAD=,F12.4)
      RETURN
      END


      SUBROUTINE VAR
      COMMON A( 25, 25),B( 25, 25),IB( 25),H( 25),AX( 25)
      COMMON X( 25),XLB( 25),XUB( 25),XI( 25),Y( 25),R( 25)
      COMMON  XSOL( 25),IRV( 25)
      COMMON MM,N,M,JS,E,IR,AD,W,Z,IM,DI,ITR,XH,AH,HH,EE,BND
      COMMON ZZ,DJS,ITER,L,IW,NODEO,INTRES,XT,XK,ID,RL,BSIGN
      COMMON  IREAD,IWRITE,IBNB


C THIS SUBROUTINE PICKS OUT THE NON-INTEGER VARIABLE IN THE
C CURRENT SOLUTION THAT WILL BE BOUNDED AND BRANCHED UPON.


      XLDU=0.0
      XLDL=0.0
      IML=0
      IML=0
   62 DO 50 IV = 1,MM
      XDU=1000000
      XDL=1000000
      IF (IB(IV)) 50,50,55
   55 IF(IRV(IB(IV)).EQ.0.AND.INTRES.EQ.1) GO TO 50
      IF(ABS(X(IB(IV))-AINT(X(IB(IV))+E))-E)50,50,60
   60 DO 61 JV=1,MM
      WRITE (IWRITE,44) B(M,JV),B(IV,JV)
   44 FORMAT (' B(M,JV)=',F14.4,',B(IV,JV)=',F9.4)
       IF(ABS(B(M,JV)).LT.E) GO TO 61
       IF(ABS(B(IV,JV)).LT.E) GO TO 61
      IF(B(IV,JV)) 64,61,63
   63 XDUJV=ABS(B(M,JV)/B(IV,JV))
      XDLJV= 0.0
      IF(XDUJV.GE.XDU) GO TO 61
      XDU=XDUJV
      WRITE (IWRITE,41)XDLJV,XDUJV,JV,IV,X(IB(IV)),IB(IV)
      GO TO 61
   64 XDLJV=ABS(B(M,JV)/B(IV,JV))
      XDUJV= 0.0
      IF(XDLJV.GE.XDL) GO TO 61
      XDL=XDLJV
      WRITE (IWRITE,41)XDLJV,XDUJV,JV,IV,X(IB(IV)),IB(IV)
   61 CONTINUE
   52 FIDLO =   X(IB(IV))- AINT(X(IB(IV)))
```

```
   53 XLDLIV=XDL*FIDLO
      XLDUIV=XDU*(1.0-FIDLO)
      IF(XLDLIV.LE.XLDL) GO TO 66
      XLDL=XLDLIV
      IML=IV
   66 IF(XLDUIV.LE.XLDU) GO TO 50
      XLDU=XLDUIV
      IMU=IV
   50 CONTINUE
      IF(XLDU-XLDL) 68,67,67
   67 IM=IMU
      BSIGN= 1.0
      GO TO 45
  100  FORMAT (F1.0)
   68 IM=IML
      BSIGN=-1.0
   45 WRITE(IWRITE,10)IM
      WRITE (6      ,10)IM
      READ (5,100)DSTOP
      IF (DSTOP.EQ.1.0)STOP
   10 FORMAT(1HO,10X, ' FROM VAR IM= ' ,I2)
      WRITE (6      ,42) BSIGN
      WRITE (IWRITE,40) FIDLO,XDU,XDL,XLDU,XLDL,IMU,
     1IML,IM
   40 FORMAT(' FIDLO=',F9.4,',XDU='F9.4,',XDL=',F9.4,
     1I4,',XLDU=',F9.4,',XLDL=',F9.4,',IMU=',I4,
     2',IML=',I4,',IM=',I4,/)
   41 FORMAT(' XDLJV=',F9.4,',XDUJV=',F9.4,',JV=',I4,',IV=',
     1I4,',X(IB(IV))=',F5.1,',IB(IV)=',I4,/)
   42 FORMAT(' BSIGN=',F9.4)
      IF(IM.EQ.0)STOP
      RETURN
      END


      SUBROUTINE REDO
      COMMON A( 25, 25),B( 25, 25),IB( 25),H( 25),AX( 25)
      COMMON X( 25),XLB( 25),XUB( 25),XI( 25),Y( 25),R( 25)
      COMMON  XSCL( 25),IRV( 25)
      COMMON MM,N,M,JS,E,IR,AD,W,Z,IM,DI,ITR,XH,AH,HH,EE,BND
      COMMON ZZ,DJS,ITER,L,IW,NODEO,INTRES,XT,XK,ID,RL,BSIGN
      COMMON  IREAD,IWRITE,IBNB

C THIS SUBROUTINE REMOVES THE LAST BOUND SO THAT BACK TRACK-
C ING CAN START, AND RESETS ALL OF THE OTHER BOUNDS.


    5 WRITE (IWRITE,14)
   14 FORMAT(1HO,10X, ' FROM REDO ')
      DO 21 K=1,N
      XUB(K)=-1.
      XLB(K)=0.
   21 CONTINUE
      LM=L-1
      IF (LM.EQ.0) GO TO  17
      DO 18 JT=1,LM
      CALL UNPAK (JT)
      K=XK
      IF (R(JT)) 19,18,20
   19 XLB(K)=XT
      WRITE(IWRITE,11)K,XLB(K)
   11 FORMAT(1HO,10X,2HK=,I2,5X,4HXLB=,F10.4)
      GO TO 18
   20 XUB(K)=XT
      WRITE(IWRITE,12)K,XUB(K)
   12 FORMAT(1H ,10X,2HK=,I2,5X,4HXUB=,F10.4)
   18 CONTINUE
   17 JT = L
      CALL UNPAK (JT)
      ID=500
      IF(BSIGN.EQ.-1.) XT=XT-1
```

```
      IF(BSIGN.EQ.+1.) XT=XT+1
      CALL PACK
      BSIGN = - BSIGN
      R(L) = -R(L)
      WRITE(IWRITE,15)L,R(L)
   15 FORMAT(1HO,10X,2HL=,I2,5X,2HR=,F14.8,'XXXXXXXXXXXXX')
    1 RETURN
      END


      SUBROUTINE PACK
      COMMON A( 25, 25),B( 25, 25),IB( 25),H( 25),AX( 25)
      COMMON X( 25),XLB( 25),XUB( 25),XI( 25),Y( 25),R( 25)
      COMMON  XSOL( 25),IRV( 25)
      COMMON MM,N,M,JS,E,IR,AD,W,Z,IM,DI,ITR,XH,AH,HH,EE,BND
      COMMON ZZ,DJS,ITER,L,IW,NODEO,INTRES,XT,XK,ID,RL,BSIGN
      COMMON  IREAD,IWRITE,IBNB

C THIS SUBROUTINE COMBINES INTO A SINGLE NUMBER THE VARIABLE
C BOUNDED, THE VALUE OF THE BOUND, AND WHETHER IT IS AN
C UPPER OR LOWER BOUND.

      IF(XT.EQ.0.) GO TO 1
      IL=1
      IT=XT
   30 IF(IT/10) 11,11,20
   20 IL=IL+1
      IT=IT/10
      GO TO 30
   11 TL=IL
      R(L) = ( XK+TL/10.+XT/10.**(TL+1.)+FLOAT(ID))*( BSIGN)
      WRITE(IWRITE,10)XT,IL,TL,L,R(L)
   10 FORMAT(1HO,10X, ' FROM PACK ' ,5X,3HXT=,F10.4,5X,3HIL=
     1,F10.4,5X,2HL=,I5,5X,2HR=,F14.8)
      GO TO 2
    1 R(L) =( XK + FLOAT(ID))*( BSIGN)
    2 RETURN
      END


      SUBROUTINE UNPAK (JT)
      COMMON A( 25, 25),B( 25, 25),IB( 25),H( 25),AX( 25)
      COMMON X( 25),XLB( 25),XUB( 25),XI( 25),Y( 25),R( 25)
      COMMON  XSOL( 25),IRV( 25)
      COMMON MM,N,M,JS,E,IR,AD,W,Z,IM,DI,ITR,XH,AH,HH,EE,BND
      COMMON ZZ,DJS,ITER,L,IW,NODEO,INTRES,XT,XK,ID,RL,BSIGN
      COMMON  IREAD,IWRITE,IBNB

C THIS SUBROUTINE RETREVIES THE INFORMATION COMBINED IN
C SUBROUTINE PACK.

      RL=ABS(R(JT))
      ID=0
      IF(RL.LT.500) GO TO 10
      RL=RL-500.
      ID=500
   10 IH=RL
      XK=IH
      TL=(RL-XK)*10.
      IL=TL
      RX=IL
      IXT=(TL-RX)*10.**RX+.5
      XT=IXT
      BSIGN = SIGN(1.0,R(L))
      RETURN
      END
```

```
      SUBROUTINE BOUND
      COMMON A( 25, 25),B( 25, 25),IB( 25),H( 25),AX( 25)
      COMMON X( 25),XLB( 25),XUB( 25),XI( 25),Y( 25),R( 25)
      COMMON  XSOL( 25),IRV( 25)
      COMMON MM,N,M,JS,E,IR,AD,W,Z,IM,DI,ITR,XH,AH,HH,EE,BND
      COMMON ZZ,DJS,ITER,L,IW,NODEO,INTRES,XT,XK,ID,RL,BSIGN
      COMMON  IREAD,IWRITE,IBNB

C THIS SUBROUTINE SETS THE VALUE OF BOUND OF THE NEXT BRANCH

      IDBND = ID
      LSET = 0
      IF(ID.EQ.500) GO TO 2
      L=L+1
      LSET = 1
   23 CALL VAR
      IF (IM.GT.0) GO TO 17
      WRITE (IWRITE,14) IM
      L = L - 1
      RETURN
   17 K=IB(IM)
      IH=X(K)*DI+.5
      XT=IH
      X(K)=XT/DI
      IH=X(K)+E
   32 XT=IH+1
      IF(BSIGN.EQ.-1.) GO TO 31
      CALL PACK
      GO TO 6
   31 Y(IM)=XT-X(K)
      X(K)=XT
      XLB(K)=XT
      IB(IM)=-IM
      M=M+1
      XI(K)=0.
      DO 1 J=1,MM
      B(M,J)=B(IM,J)
    1 B(IM,J)=-B(IM,J)
      H(IM)=-H(IM)
      W=Y(IM)
      IW=0
      WRITE(IWRITE,16) W
      IF(IDBND.EQ.500)RETURN
      XK=K
      CALL PACK
      GO TO 3
    2 K=XK
      IF(ABS(XT-X(K))-E) 21,21,22
   21 ID=0
      IF(BSIGN.LT.0.0) GO TO 30
      XUB(K)=XT
      GO TO 23
   30 XLB(K)=XT
      GO TO 23
   22 IF(XT.GE.X(K).AND.BSIGN.EQ.+1.) GO TO 21
      IF(XT.LE.X(K).AND.BSIGN.EQ.-1.) GO TO 21
      DO 5 I=1,M
      IF(IB(I).EQ.K)GO TO 4
      GO TO 5
    4 IM = I
      WRITE(IWRITE,14)IM
   14 FORMAT(1H0,10X, ' FROM BOUND IM= ' ,I5)
      GO TO 8
    5 CONTINUE
      WRITE(IWRITE,15)
   15 FORMAT(1H0,10X, ' ERROR ' )
```

```
    8 ID = 0
      IF (BSIGN.LT.0.0) GO TO 31
    6 Y(IM)=X(K)-XT
      X(K)=XT
      XUB(K)=XT
      WRITE(IWRITE,10)K,X(K)
   10 FORMAT(1HO,10X, ' FROM BOUND ' ,5X,2HK=,I2,5X,2HX=,F10
      IB(IM)=-IM
      M=M+1
      XI(K)=0.
      DO 7 J=1,MM
    7 B(M,J)=-B(IM,J)
      W=Y(IM)
      IW=0
      WRITE(IWRITE,16)W
   16 FORMAT(1HO,10X, ' FROM BOUND W= ' ,F10.8)
      IF(LSET.EQ.1)RETURN
      L = L + 1
      XK = K
      CALL PACK
    3 RETURN
      END


      SUBROUTINE UNBND
      COMMON A( 25, 25),B( 25, 25),IB( 25),H( 25),AX( 25)
      COMMON X( 25),XLB( 25),XUB( 25),XI( 25),Y( 25),R( 25)
      COMMON  XSOL( 25),IRV( 25)
      COMMON MM,N,M,JS,E,IR,AD,W,Z,IM,DI,ITR,XH,AH,HH,EE,BND
      COMMON ZZ,DJS,ITER,L,IW,NODEO,INTRES,XT,XK,ID,RL,BSIGN
      COMMON  IREAD,IWRITE,IBNB

C THIS SUBROUTINE RESETS ALL THE BOUNDS AFTER EACH BACKTRACK
C AND DETERMINES WHICH NODE TO BACKTRACK TO.

    6 IF(ABS(R(L))-500.0) 1,1,2
    1 CALL REDO
      ID=500
      RETURN
    2 IF(L.EQ.1) CALL REDO
      IF(L.EQ.1)  RETURN
      L = L - 1
      GO TO 6
      END


      SUBROUTINE ELEM
      COMMON A( 25, 25),B( 25, 25),IB( 25),H( 25),AX( 25)
      COMMON X( 25),XLB( 25),XUB( 25),XI( 25),Y( 25),R( 25)
      COMMON  XSOL( 25),IRV( 25)
      COMMON MM,N,M,JS,E,IR,AD,W,Z,IM,DI,ITR,XH,AH,HH,EE,BND
      COMMON ZZ,DJS,ITER,L,IW,NODEO,INTRES,XT,XK,ID,RL,BSIGN
      COMMON  IREAD,IWRITE,IBNB

C THIS IS THE SUBPROGRAM WHICH OBTAINS THE PIVOT ELEMENT
C FOR THE GIVEN COLUMN DETERMINED EITHER IN 'LP' OR
C IN 'BOUND'.
C IT GIVES WHICH VARIABLE WILL LEAVE THE BASIS.

      IR=0
      CALL AXE
      EE=E
      WRITE(IWRITE,43)E
   43 FORMAT(1HO,10X,12HFROM ELEM E=,F10.4)
      IF(BND.EQ.0.)GO TO 12
      IF(XI(JS).EQ.1.)GO TO 1
      GO TO 2
    1 XH=XUB(JS)-X(JS)
```

```
      IF(XUB(JS).EQ.-1.)XH=1000000.
      GO TO 12
    2 XH=X(JS)-XLB(JS)
   12 DO 25 I=1,MM
      IF(BND.GT.0.)GO TO 810
      IF(AX(I).LE.E)GO TO 25
      AH=AX(I)
      HH=H(I)
      IF(IR)25,811,812
  811 IR=I
      XH=HH/AH
      EE=E/AH
      GO TO 25
  812 CALL MIN (I)
      GO TO 25
  810 IF(ABS(AX(I)).LT.E)GO TO 25
      IF(XI(JS).EQ.1.)GO TO 8
C DECREASE.
      IF(IB(I))3,25,4
    4 K=IB(I)
      IF(AX(I))5,25,6
    6 IF(XUB(K).EQ.-1.)GO TO 25
      HH=XUB(K)-X(K)
      AH=-AX(I)*XI(JS)
      CALL MIN (I)
      GO TO 25
    5 HH=X(K)-XLB(K)
      AH=AX(I)*XI(JS)
      CALL MIN (I)
      GO TO 25
    3 IF(AX(I))15,25,16
   16 HH=Y(I)
      AH=-AX(I)*XI(JS)
      CALL MIN (I)
      GO TO 25
   15 HH=Y(I)
      AH=AX(I)*XI(JS)
      CALL MIN (I)
      GO TO 25
C INCREASE.
    8 IF(IB(I))30,25,40
   40 K=IB(I)
      IF(AX(I))6,25,5
   30 IF(AX(I))16,25,15
   25 CONTINUE
      RETURN
      END


      SUBROUTINE MIN (I)
      COMMON A( 25, 25),B( 25, 25),IB( 25),H( 25),AX( 25)
      COMMON X( 25),XLB( 25),XUB( 25),XI( 25),Y( 25),R( 25)
      COMMON  XSOL( 25),IRV( 25)
      COMMON MM,N,M,JS,E,IR,AD,W,Z,IM,DI,ITR,XH,AH,HH,EE,BND
      COMMON ZZ,CJS,ITER,L,IW,NODEO,INTRES,XT,XK,ID,RL,BSIGN
      COMMON  IREAD,IWRITE,IBNB

C THIS IS THE SUBROUTINE USED TO CALCULATE THE MINIMUM THETA

      IF(ABS(HH-AH*XH)-EE)70,70,2
    2 IF(HH-AH*XH)24,70,25
   24 XH=HH/AH
    3 IR=I
      EE=E/AH
      GO TO 25
   70 IF(IR.EQ.0) GO TO 25
      IF(IB(I))74,25,73
   73 IF(IR.EQ.0) GO TO 25
      IF (IB(IR)) 25,25,4
   74 IF(IR.EQ.0) GO TO 3
```

```
      IF (IB(IR)) 4,25,3
    4 IF(ABS(AX(IR))-ABS(AX(I)))25,25,3
   25 RETURN
      END
```

ASSEMBLY LANGUAGE TIMING ROUTINE

```
SETIME   START
         SAVE   (14,12)
         LR     12,15
         USING SETIME,12
         ST     13,TEMP
         LA     13,SAVE
         STIMER TASK,FIXUP,TUINTVL=TIME
         L      13,TEMP
         RETURN (14,12),T
         DROP   12
FIXUP    SAVE   (14,12)
         RETURN (14,12)
         ENTRY WATIME
WATIME   SAVE   (14,12)
         LR     12,15
         USING WATIME,12
         L      2,0(1)
         ST     13,TEMP
         LA     13,SAVE
         TTIMER CANCEL
         L      1,TIME
         SR     1,0
         ST     1,0(2)
         L      13,TEMP
         RETURN (14,12),T
SAVE     DS     18F
TEMP     DS     F
TIME     DC     X'08000000'
         END
```

BIBLIOGRAPHY

1. Balinski, M.L., "Integer Programming: Methods, Uses Computation," Management Science, Vol. 12(1965), pp. 253-313.

2. Bellmore, M. and Nemhauser, B.L., "The Traveling Salesman Problem: A Survey," Operations Research, Vol. 16(1968), pp. 538-558.

3. Dantzig, G. D., Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey (1963).

4. Driebeck, N.J., "A Method for the Solution of Mixed Integer and Nonlinear Programming Problems by Linear Programmnng Methods," International Symposium on Mathematical Programming, London(1964). (The methods are more readily available in Beale, E.M.L., Mathematical Programming in Practice, Sir Issac Pitman and Sons, Ltd., London(1968).

5. Greenberg,H., ON Solving Integer Programs, Naval Postgraduate School, NPS55GD8051A, Monterey, California (1968).

6. Hadley, G., Linear Programming, Adison-Wesley Publishing Company, Inc., Palo Alto, California(1962).

7. Haldi, John, "25 Integer Programming Test Problems," Working Paper No. 43, Graduate School of Business, Stanford University, December 1964.

8. Krolak, P.D., "Computational Results of an Integer Programming Algorithm," Operations Research, Vol. 17 (1969), pp. 743-748.

9. Land, A.H. and Doig, A.G., "An Automatic Method of Solving Discrete Programming Problems," Econometrica, Vol. 28(1960), pp. 497-520.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	20
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Director, Systems Analysis Division (OP 96) Office of the Chief of Naval Operations Washington, D. C. 20315	1
4.	Department of the Army Civil Sshools Branch, OPO, OPD Washington, D. C. 20315	1
5.	Professor Harold Greenberg, Code 55GD Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	1
6.	Captain Joseph B. Missal 1064 Haviland Terrace Seaside, California 93955	1
7.	Department of Operations Analysis, Code 55 Naval Postgraduate School Monterey, California 93940	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
A Branch and Bound Algorithm for Integer and Mixed-Integer Linear Programs

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Master's Thesis; October 1969

5. AUTHOR(S) *(First name, middle initial, last name)*
Joseph Bannan Missal

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
October 1969	46	9

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

The algorithm presented is an extension of the Land and Doig branch and bound method combined with the branch selection techniques presented by Beale and Small to solve integer or mixed-integer linear programs. The algorithm obtains the solution by solving a linear program with upper and/or lower bounds on selected branch variables. By systematically changing these bounds, and maintaining only the current canonical form, the solution is assured using a minimum of excess computer storage above that required to solve the linear programming problem. Thus the problem can be solved entirely within the computer core, and the problem converges to the solution faster than most other general integer linear programming algorithms.

DD FORM 1 NOV 65 1473 (PAGE 1)
S/N 0101-807-6811

Security Classification

A-31408

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Integer						
Integer Programming						
Mathematical Programming						
Mixed-Integer						
Mixed-Integer Programming						
Mixed Integer						
Simplex						